CMSC 351 - Spring, 2011 - Project This is an INDIVIDUAL assignment - NO discussion with others, except for TA, Dr. Hugue, or anyone designated by Dr. Hugue. Due 11:49pm, April, 22nd.

For this project you need to implement a parameterized version of the "Median-of-Medians" styled Select(list,i) algorithm. You will generate the recurrence relation based on the parameters, and will chart actual performance of the algorithm empirically.

Implement **Select(Comparable[] list, int pos, int colSize, int colMed)** in **CMSC351P1.java**.

- **list**: list of values of which to find a specified position
- **pos**: the specified position
- **colSize**: the size of the "columns" that we create in the first stage
- **colMed**: the position in those "columns" that we use as the "MedX"

For this project, when the size of the list is less than colSize^2 you should go ahead and sort the list (you decide what sorting algorithm to use, but you must count the comparisons it does on the data) and then pick the desired element. You can also find the colMed within the columns by sorting (you again decide what sorting algorithm to use, maybe the same as the previous step but maybe different) and then picking the specified element (again counting the comparisons done).

For median finding, this would be called **Select(list,list.length/2);**. This is how it is called in the posted P1driver.java file.

We will **not** assume that list elements are unique. You should assume that any test file can include multiple occurrences of some keys. While this might make it easier for you to generate some of the large inputs sets that you should use to test you code, do not limit yourself to randomly generated data, either.

For simplicity of debugging and testing, we are using integers for nearly all of the test files used to evaluate your code. However, your code should also work (for example) on strings, which is why Comparable is the specified data type, and we may have one or two test runs which violate the 'all integer' assumption.

This project will have two files. We provide a P1driver.java that you will need to modify to print the recurrence relation based on the input values for colSize and colMed, and a file with a simple skeleton for your Select() method, CMSC351P1.java. You might also want to write an alternate driver to make it easier to run your algorithm on many data sets and collect your data more easily. You will also want to write an alternate driver to test that your Select() method works for things other than median finding. You can add any helper methods to this class that you wish. Within your code, anywhere a **comparison to a data value takes place**, you must increment the counter called **comps** in the **CMSC351P1** class. All comparisons to the data array must be counted for full credit.

You need to run tests with different values of colSize and colMed to determine what a good values for these might be in practice. One example of what you could do is run the same set of large input files with different values sent to the algorithm to see when the number of comparisons performed is minimized. You are required to submit a **PDF** file named **README.pdf** with a description of the technique(s) you used to pick your colSize and colMed value, and some charts of your data (for example, the number of comparisons -vs- colSize -vs- colMed) and a brief explanation of **why** your data leads to your choice for colSize and/or colMed. To create a PDF file from a word processing document, you can install the free CutePDF Writer virtual printer on a

Windows machine, or print to PDF on a Mac.

You will submit a ZIP file containing *your* **CMSC351P1.java** and *your* **P1driver.java** which prints the recurrence relation, and your **README.pdf** files using the submit.cs.umd.edu server. Grading will be based on (a) your method implementing Select(), (b) your written description and discussion of the techniques you used to pick your ideal colSize and colMed values, and (c) whether or not your Select() returns the correct value when an arbitrary element of the list is specified, not just the median--so be sure to verify that your algorithm works when any rank in the list is specified.